The Fibonacci Sequence

Example: http://goo.gl/dcaf0
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, ...

Example: http://goo.gl/dcaf0
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, ...

def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1  # First two Fibonacci numbers
    k = 2               # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr

Example: http://goo.gl/dcaf0
The Fibonacci Sequence

```
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1  # First two Fibonacci numbers
    k = 2              # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

0, 1, 1, 2, 3, 5, 8, 13, ...

Example: http://goo.gl/dcaf0
The Fibonacci Sequence

def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1  # First two Fibonacci numbers
    k = 2              # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr

Example: http://goo.gl/dcaf0
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr

Example: http://goo.gl/dcaf0
### The Fibonacci Sequence

**def** fib(n):

"""Compute the nth Fibonacci number, for n >= 2."""

pred, curr = 0, 1  # First two Fibonacci numbers
k = 2               # Tracks which Fib number is curr

while k < n:
    pred, curr = curr, pred + curr
    k = k + 1

return curr

Example: [http://goo.gl/dcaf0](http://goo.gl/dcaf0)
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

Example: [http://goo.gl/dcaf0](http://goo.gl/dcaf0)
The Fibonacci Sequence

```
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1  # First two Fibonacci numbers
    k = 2              # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

0, 1, 1, 2, 3, 5, 8, 13, ...

Example: http://goo.gl/dcaf0
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1  # First two Fibonacci numbers
    k = 2              # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

Example: [http://goo.gl/dcaf0](http://goo.gl/dcaf0)
Practical Guidance: the Art of the Function
Practical Guidance: the Art of the Function

\[
\begin{bmatrix}
  r_0 \\
  \theta \\
  z
\end{bmatrix} = \begin{bmatrix}
  3 + \sin t + \cos u \\
  2t \\
  \sin u + 2 \cos t
\end{bmatrix}, \quad t = 0 \ldots 2\pi, \quad u = 0 \ldots 2\pi
\]
Give each function exactly one job.
Practical Guidance: the Art of the Function

Give each function exactly one job.
Practical Guidance: the Art of the Function

Give each function exactly one job.

vs

Don’t repeat yourself (DRY). Implement a computational process just once, but execute it many times.

\[
\begin{align*}
\begin{bmatrix}
 r_0 \\
 \theta \\
 z \\
\end{bmatrix} &= \begin{bmatrix} 3 + \sin t + \cos u \\ 2t \\ \sin u + 2 \cos t \end{bmatrix}, \\
& \text{for } t = 0 \ldots 2\pi, \ u = 0 \ldots 2\pi
\end{align*}
\]
Practical Guidance: the Art of the Function

Give each function exactly one job.

vs

Don’t repeat yourself (DRY). Implement a computational process just once, but execute it many times.

\[
\begin{bmatrix}
\theta \\
3+\sin t+\cos u \\
\sin u+2\cos t
\end{bmatrix}, t=0\ldots2\pi, u=0\ldots2\pi
\]
Practical Guidance: the Art of the Function

Give each function exactly one job.

Don’t repeat yourself (DRY). Implement a computational process just once, but execute it many times.

Define functions generally.

\[
\begin{bmatrix}
 r_0 \\
 \theta \\
 z
\end{bmatrix} = \begin{bmatrix}
 3 + \sin(t) + \cos(u) \\
 2t \\
 \sin(u) + 2\cos(t)
\end{bmatrix}, \quad t=0\ldots2\pi, \quad u=0\ldots2\pi
\]
Practical Guidance: the Art of the Function

Give each function exactly one job.

Don’t repeat yourself (DRY). Implement a computational process just once, but execute it many times.

Define functions generally.
Generalizing Patterns with Arguments
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape: 

Area: $r^2$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
- Square
- Circle
- Hexagon

Area:
- Square: \( r^2 \)
- Circle: \( \pi \cdot r^2 \)
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:

\[ \pi \cdot r^2 \]

\[ \frac{3\sqrt{3}}{2} \cdot r^2 \]
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape: 

Area: 

\[ \text{Square: } 1 \cdot r^2 \]

\[ \text{Circle: } \pi \cdot r^2 \]

\[ \text{Hexagon: } \frac{3\sqrt{3}}{2} \cdot r^2 \]
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
- Square
- Circle
- Hexagon

Area:
- Square: $1 \cdot r^2$
- Circle: $\pi \cdot r^2$
- Hexagon: $\frac{3\sqrt{3}}{2} \cdot r^2$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

**Shape:**
- Square: $1 \cdot r^2$
- Circle: $\pi \cdot r^2$
- Hexagon: $\frac{3\sqrt{3}}{2} \cdot r^2$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:  
- Square: $r^2$
- Circle: $\pi r^2$
- Hexagon: $\frac{3\sqrt{3}}{2} r^2$

Area:  
- Square: $1 \cdot r^2$
- Circle: $\pi \cdot r^2$
- Hexagon: $\frac{3\sqrt{3}}{2} \cdot r^2$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:

Finding common structure allows for shared implementation
Generalizing Over Computational Processes
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.

\[ \sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15 \]

\[ \sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225 \]

\[ \sum_{k=1}^{5} \frac{8}{(4k - 3) \cdot (4k - 1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04 \]
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.

\[
\sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15
\]

\[
\sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225
\]

\[
\sum_{k=1}^{5} \frac{8}{(4k - 3) \cdot (4k - 1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04
\]
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.

\[
\sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15
\]

\[
\sum_{k=1}^{5} (k^3) = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225
\]

\[
\sum_{k=1}^{5} \frac{8}{(4k - 3) \cdot (4k - 1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04
\]
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.

\[
\sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15
\]

\[
\sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225
\]

\[
\sum_{k=1}^{5} \frac{8}{(4k - 3) \cdot (4k - 1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04
\]
Summation Example

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."

    >>> summation(5, cube)
    225
    """
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
```

# Local function definitions; returning functions

def make_adder(n):
    """Return a function that takes one argument k and returns k + n."

    >>> add_three = make_adder(3)
    >>> add_three(4)
    7
    """
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    """Return a function that composes f and g.
    f, g −− functions of a single argument"

def h(x):
    return f(g(x))
    return h
```
Summation Example

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."

    >>> summation(5, cube)
    225
    """
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
```

Function of a single argument (not called term)
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
"""
Summation Example

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."

    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
"""
```

Function of a single argument (not called term)

A formal parameter that will be bound to a function

The function bound to term gets called here
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>> sumation(5, cube)
225
"""

The cube function is passed as an argument value

The function bound to term gets called here

Function of a single argument (not called term)

A formal parameter that will be bound to a function
Summation Example

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
"""
```

Function of a single argument (not called term)

A formal parameter that will be bound to a function

The cube function is passed as an argument value

The function bound to term gets called here

$0 + 1^3 + 2^3 + 3^3 + 4^3 + 5^5$
Locally Defined Functions
Locally Defined Functions

Functions defined within other function bodies are bound to names in the local frame.
Locally Defined Functions

Functions defined within other function bodies are bound to names in the local frame

```python
def make_adder(n):
    """Return a function that takes one argument k and returns k + n."

    >>> add_three = make_adder(3)
    >>> add_three(4)
    7
    """
    def adder(k):
        return k + n
    return adder
```
Locally Defined Functions

Functions defined within other function bodies are bound to names in the local frame

A function that returns a function

```python
def make_adder(n):
    # Return a function that takes one argument k and returns k + n.
    def adder(k):
        return k + n
    return adder
```

```python
>>> add_three = make_adder(3)
>>> add_three(4)
7

def adder(k):
    return k + n
return adder
```
Locally Defined Functions

Functions defined within other function bodies are bound to names in the local frame.

A function that returns a function

```python
def make_adder(n):
    """
    Return a function that takes one argument k and returns k + n.
    """
    def adder(k):
        return k + n
    return adder

>>> add_three = make_adder(3)
>>> add_three(4)
7

The name add_three is bound to a function
```
Locally Defined Functions

Functions defined within other function bodies are bound to names in the local frame.

A function that returns a function

```python
def make_adder(n):
    """Return a function that takes one argument k and returns k + n."
    return lambda k: k + n

>>> add_three = make_adder(3)
>>> add_three(4)
7
```

The name `add_three` is bound to a function

A local def statement

```python
def adder(k):
    return k + n
return adder
```
Locally Defined Functions

Functions defined within other function bodies are bound to names in the local frame

A function that returns a function

```
def make_adder(n):
    """Return a function that takes one argument k and returns k + n."""
    def adder(k):
        return k + n
    return adder
```

The name add_three is bound to a function

A local def statement

Can refer to names in the enclosing function
Call Expressions as Operator Expressions

```python
def make_adder(n):
    def adder(k):
        return k + n
    return adder

make_adder(1)(2)
```
Call Expressions as Operator Expressions

```python
def make_adder(n):
    def adder(k):
        return k + n
    return adder

make_adder(1)(2)
```

```python
def make_adder(n):
    def adder(k):
        return k + n
    return adder

make_adder(1)(2)
```
Call Expressions as Operator Expressions

\[
\text{make_adder}(1)(2)
\]

\[
\text{make_adder}(1) \ ( \ 2 \ )
\]
Call Expressions as Operator Expressions

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

make_adder(1)(2)  # make_adder(1)     (        2         )
```
Call Expressions as Operator Expressions

```python
def make_adder(n):
    def adder(k):
        return k + n
    return adder

make_adder(1)(2)
```

An expression that evaluates to a function
Call Expressions as Operator Expressions

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder
make_adder(1)(2)
```

An expression that evaluates to a function

An expression that evaluates to any value

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder
make_adder(1)(2)
```
The Purpose of Higher-Order Functions
The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.
The Purpose of Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value.
The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Higher-order function: A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:
The Purpose of Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

- Express general methods of computation
The Purpose of Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

- Express general methods of computation
- Remove repetition from programs
The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Higher-order function: A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions
Pig Introduction

(Demo)